Space- and Time-Efficient Algorithm for Maintaining Dense Subgraphs on One-Pass Dynamic Streams*

Sayan Bhattacharya
Institute of Mathematical Sciences, Chennai
bsayan@imsc.res.in

Monika Henzinger
University of Vienna
monika.henzinger@univie.ac.at

Danupon Nanongkai
KTH Royal Institute of Technology
danupon@gmail.com

Charalampos E. Tsourakakis
Harvard University
babis@seas.harvard.edu

ABSTRACT

While in many graph mining applications it is crucial to handle a stream of updates efficiently in terms of *both* time and space, not much was known about achieving such type of algorithm. In this paper we study this issue for a problem which lies at the core of many graph mining applications called *densest subgraph problem*. We develop an algorithm that achieves time- and space-efficiency for this problem simultaneously. It is one of the first of its kind for graph problems to the best of our knowledge.

Given an input graph, the densest subgraph is the subgraph that maximizes the ratio between the number of edges and the number of nodes. For any $\epsilon > 0$, our algorithm can, with high probability, maintain a $(4+\epsilon)$-approximate solution under edge insertions and deletions using $\tilde{O}(n)$ space and $\tilde{O}(1)$ amortized time per update; here, n is the number of nodes in the graph and $\tilde{O}$ hides the $O(\text{poly}\log_{1+\epsilon} n)$ term. The approximation ratio can be improved to $(2+\epsilon)$ with more time. It can be extended to a $(2+\epsilon)$-approximation sublinear-time algorithm and a distributed-streaming algorithm. Our algorithm is the first streaming algorithm that can maintain the densest subgraph in *one pass*. Prior to this, no algorithm could do so even in the special case of an incremental stream and even when there is no time restriction. The previously best algorithm in this setting required $O(\log n)$ passes [6]. The space required by our algorithm is tight up to a polylogarithmic factor.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*

General Terms

Algorithms, Theory

Keywords

Dynamic graph algorithms; streaming algorithms; dense subgraphs

*The full version of this paper is available as [7] at `http://arxiv.org/abs/1504.02268`

1. INTRODUCTION

In analyzing large-scale rapidly-changing graphs, it is crucial that algorithms must use small space and adapt to the change quickly. This is the main subject of interest in at least two areas, namely *data streams* and *dynamic algorithms*. In the context of graph problems, both areas are interested in maintaining some graph property, such as connectivity or distances, for graphs undergoing a stream of edge insertions and deletions. This is known as the (one-pass) *dynamic semi-streaming* model in the data streams community, and as the *fully-dynamic* model in the dynamic algorithm community.

The two areas have been actively studied since at least the early 80s (e.g. [17, 32]) and have produced several sophisticated techniques for achieving time and space efficiency. In dynamic algorithms, where the primary concern is *time*, the heavy use of *amortized analysis* has led to several extremely fast algorithms that can process updates and answer questions in a poly-logarithmic amortized time. In data streams, where the primary concern is *space*, the heavy use of *sampling* techniques to maintain small *sketches* has led to algorithms that require space significantly less than the input size; in particular, for dynamic graph streams the result by Ahn, Guha, and McGregor [1] has demonstrated the power of linear graph sketches in the dynamic model, and initiated an extensive study of dynamic graph streams (e.g. [25, 26, 1, 2, 3]). Despite numerous successes in these two areas, we are not aware of many results that combine techniques from *both* areas to achieve time- and space-efficiency *simultaneously* in dynamic graph streams. A notable exception we are aware of is the connectivity problem, where one can combine the space-efficient streaming algorithm of Ahn et al. [2] with the fully-dynamic algorithm of Kapron et al. [27][1].

In this paper, we study this issue for the *densest subgraph* problem. For any unweighted undirected graph G, the density of G is defined as $\rho(G) = |E(G)|/|V(G)|$. The dens-

[1]We thank Valerie King (private communication) for pointing out this fact.

est subgraph of G is the subgraph H that maximizes $\rho(H)$, and we denote the density of such subgraph by $\rho^*(G) = \max_{H \subseteq G} \rho(H)$. For any $\gamma \geq 1$ and ρ', we say that ρ' is an γ-*approximate value* of $\rho^*(G)$ if $\rho^*(G)/\gamma \leq \rho' \leq \rho^*(G)$. The (static) densest subgraph problem is to compute or approximate ρ^* and the corresponding subgraph. Throughout, we use n and m to denote the number of nodes and edges in the input graph, respectively.

This problem and its variants have been intensively studied in practical areas as it is an important primitive in analyzing massive graphs. Its applications range from identifying dense communities in social networks (e.g. [14, 41]), link spam detection (e.g. [18]) and finding stories and events (e.g. [4]); for many more applications of this problem see, e.g., [6, 29, 40, 39]. Goldberg [20] was one of the first to study this problem although the notion of graph density has been around much earlier (e.g. [28, Chapter 4]). His algorithm can solve this problem in polynomial time by using $O(\log n)$ flow computations. Later Gallo, Grigoriadis and Tarjan slightly improved the running time using parametric maximum flow computation. These algorithms are, however, not very practical, and an algorithm that is more popular in practice is an $O(m)$-time $O(m)$-space 2-approximation algorithm of Charikar [10]. However, as mentioned earlier, graphs arising in modern applications are huge and keep changing. This algorithm is not suitable to handle such graphs. Consider, for example, an application of detecting a dense community in social networks. Since people can make new friends as well as "unfriend" their old friends, the algorithm must be able to process these updates efficiently. With this motivation, it is natural to consider the dynamic version of this problem. To be precise, we define the problem following the dynamic algorithms literature as follows. We say that an algorithm is a *fully-dynamic γ-approximation* algorithm for the densest subgraph problem if it can process the following operations.

- INITIALIZE(n): Initialize the algorithm with an empty n-node graph.
- INSERT(u, v): Insert edge (u, v) to the graph.
- DELETE(u, v): Delete edge (u, v) from the graph.
- QUERYVALUE: Output a γ-approximate value of $\rho^*(G)$.[2]

The *space complexity* of an algorithm is defined to be the space needed in the worst case. We define *time complexity* separately for each type of operations: Time for the INITIALIZE operation is called *preprocessing time*, time to process each INSERT and DELETE operation is called *update time*, time for answering each QUERY operation is called *query time*. For any τ, we say that an algorithm has an *amortized* update time τ if the total time it needs to process any k insert and delete operations is at most $k\tau$.

Our Results. Our main result is an efficient $(4+\epsilon)$-approximation algorithm for this problem, formally stated as follows. For every integer $t \geq 0$, let $G^{(t)} = (V, E^{(t)})$ be the state of the input graph $G = (V, E)$ just after we have processed the first t updates in the dynamic stream, and define $m^{(t)} \leftarrow |E^{(t)}|$. We assume that $m^{(0)} = 0$ and $m^{(t)} > 0$ for all $t \geq 1$. Let $\text{OPT}^{(t)}$ denote the density of the densest subgraph in $G^{(t)}$.

[2]We note that we can also quickly return the subgraph whose density γ-approximates $\rho^*(G)$.

THEOREM 1.1. *Fix some small constant $\epsilon \in (0, 1)$, a constant $\lambda > 1$, and let $T = \lceil n^\lambda \rceil$. There is an algorithm that processes the first T updates in the dynamic stream using $\tilde{O}(n)$ space and maintains a value $\text{OUTPUT}^{(t)}$ at each $t \in [T]$. The algorithm gives the following guarantees with high probability: We have $\text{OPT}^{(t)}/(4 + O(\epsilon)) \leq \text{OUTPUT}^{(t)} \leq \text{OPT}^{(t)}$ for all $t \in [T]$. Further, the total amount of computation performed while processing the first T updates in the dynamic stream is $O(T \, poly \log n)$.*

We note that our algorithm can be easily extended to output the set of nodes in the subgraph whose density $(4+\epsilon)$-approximates $\rho^*(G)$ using $O(1)$ time per node. As a by product of our techniques, we obtain some additional results. See the full version of our work [7] for details.

• **A $(2+\epsilon)$-approximation one-pass dynamic semi-streaming algorithm:** This follows from the fact that with the same space, preprocessing time, and update time, and an additional $\tilde{O}(n)$ query time, our main algorithm can output a $(2+\epsilon)$-approximate solution. See Section 3.

• **Sublinear-time algorithm:** We show that Charikar's linear-time linear-space algorithm can be improved further! In particular, if the graph is represented by an incident list (this is a standard representation [11, 19]), our algorithm needs to read only $\tilde{O}(n)$ edges in the graph (even if the graph is dense) and requires $\tilde{O}(n)$ time to output a $(2+\epsilon)$-approximate solution. We also provide a lower bound that matches this running time up to a poly-logarithmic factor. See [7, Appendix A] for details.

• **Distributed streaming algorithm:** In the distributed streaming setting with k sites as defined in [12], we can compute a $(2+\epsilon)$-approximate solution with $\tilde{O}(k+n)$ communication by employing the algorithm of Cormode et al. [12]. [7, Appendix B] for details.

To the best of our knowledge, our main algorithm is the first dynamic graph algorithm that requires $\tilde{O}(n)$ space (in other words, a dynamic semi-streaming algorithm) and at the same time can quickly process each update and answer each query. Previously, there was no space-efficient algorithm known for this problem, even when time efficiency is not a concern, and even in the conventional streaming model where there are only edge insertions. In this insertion-only model, Bahmani, Kumar, and Vassilvitskii [6] provided a deterministic $(2+\epsilon)$-approximation $O(n)$-space algorithm. Their algorithm needs $O(\log_{1+\epsilon} n)$ passes; i.e., it has to read through the sequence of edge insertions $O(\log_{1+\epsilon} n)$ times. (Their algorithm was also extended to a MapReduce algorithm, which was later improved by [5].) Our $(2+\epsilon)$-approximation dynamic streaming algorithm improves this algorithm in terms of the number of passes. The space usage of our dynamic algorithms matches the lower bound provided by [6, Lemma 7] up to a polylogarithmic factor.

We note that while in some settings it is reasonable to compute the solution at the end of the stream or even make multiple passes (e.g. when the graph is kept on an external memory), and thus our and Bahmani et al's $(2+\epsilon)$-approximation algorithms are sufficient in these settings, there are many natural settings where the stream keeps changing, e.g. social networks where users keep making new friends and disconnecting from old friends. In the latter case our main algorithm is necessary since it can quickly prepare to answer the densest subgraph query after every update.

Another related result in the streaming setting is by Ahn et al. [2] which approximates the fraction of some dense subgraphs such as a small clique in dynamic streams. This algorithm does not solve the densest subgraph problem but might be useful for similar applications.

Not much was known about time-efficient algorithm for this problem even when space efficiency is not a concern. One possibility is to adapt dynamic algorithms for the related problem called *dynamic arboricity*. The arboricity of a graph G is $\alpha(G) = \max_{U \subseteq V(G)} |E(U)|/(|U|-1)$ where $E(U)$ is the subgraph of G induced by U. Observe that $\rho^*(G) \leq \alpha(G) \leq 2\rho^*(G)$. Thus, a γ-approximation for the arboricity problem will be a (2γ)-approximation algorithm. In particular, we can use the 4-approximation algorithm of Brodal and Fagerberg [8] to maintain an 8-approximate solution to the densest subgraph problem in $\tilde{O}(1)$ amortized update time. (With a little more thought, one can in fact improve the approximation ratio to 6.) In the paper that appeared at about the same time as this paper, Epasto et al. [15] presented a $(2+\epsilon)$-approximation algorithm which can handle arbitrary edge insertions and random edge deletions.

Overview. An intuitive way to combine techniques from data streams and dynamic algorithms for any problem is to run the dynamic algorithm using the sketch produced by the streaming algorithm as an input. This idea does not work straightforwardly. The first obvious issue is that the streaming algorithm might take excessively long time to maintain its sketch and the dynamic algorithm might require an excessively large additional space. A more subtle issue is that the sketch might need to be processed in a specific way to recover a solution, and the dynamic algorithm might not be able to facilitate this. As an extreme example, imagine that the sketch for our problem is not even a graph; in this case, we cannot even feed this sketch to a dynamic algorithm as an input.

The key idea that allows us to get around this difficulty is to develop streaming and dynamic algorithms based on the same structure called *(α, d, L)-decomposition*. This structure is an extension of a concept called *d-core*, which was studied in graph theory since at least the 60s (e.g., [16, 30, 38]) and has played an important role in the studies of the densest subgraph problem (e.g., [6, 37]). The d-core of a graph is its (unique) largest induced subgraph with every node having degree at least d. It can be computed by repeatedly removing nodes of degree less than d from the graph, and can be used to 2-approximate the densest subgraph. Our (α, d, L)-decomposition with parameter $\alpha \geq 1$ is an approximate version of this process where we repeatedly remove nodes of degree "approximately" less than d: in this decomposition we must remove all nodes of degree less than d and are allowed to remove *some* nodes of degree between d and αd. We will repeat this process for L iterations. Note that the (α, d, L)-decomposition of a graph is not unique. However, for $L = O(\log_{1+\epsilon} n)$, an (α, d, L)-decomposition can be use to $2\alpha(1+\epsilon)^2$-approximate the densest subgraph. We explain this concept in detail in Section 2.

We show that this concept can be used to obtain an approximate solution to the densest subgraph problem and leads to both a streaming algorithm with a small sketch and a dynamic algorithm with small amortized update time. In particular, it is intuitive that to check if a node has degree approximately d, it suffices to sample every edge with probability roughly $1/d$. The value of d that we are interested in approximately ρ^*, which can be shown to be roughly the same as the average degree of the graph. Using this fact, it follows almost immediately that we only have to sample $\tilde{O}(n)$ edges. Thus, to repeatedly remove nodes for L iterations, we will need to sample $\tilde{O}(Ln) = \tilde{O}(n)$ edges (we need to sample a new set of edges in every iteration to avoid dependencies).

We turn the (α, d, L)-decomposition concept into a dynamic algorithm by dynamically maintaining the sets of nodes removed in each of the L iterations, called *levels*. Since the (α, d, L)-decomposition gives us a choice whether to keep or remove each node of degree between d and αd, we can save time needed to maintain this decomposition by moving nodes between levels *only when it is necessary*. If we allow α to be large enough, nodes will not be moved often and we can obtain a small amortized update time; in particular, it can be shown that the amortized update time is $\tilde{O}(1)$ if $\alpha \geq 2+\epsilon$. In analyzing an amortized time, it is usually tricky to come up with the right *potential function* that can keep track of the cost of moving nodes between levels, which is not frequent but expensive. In case of our algorithm, we define two potential functions for our amortized analysis, one on nodes and one on edges. (For intuition, we provide an analysis for the simpler case where we run this dynamic algorithm directly on the input graph in [7].)

Our goal is to run the dynamic algorithm on top of the sketch maintained by our streaming algorithm in order to maintain the (α, d, L)-decomposition. To do this, there are a few issues we have to deal with that makes the analysis rather complicated: Recall that in the sketch we maintain L sets of sampled edges, and for each of the L iterations we use different such sets to determine which nodes to remove. This causes the potential functions and its analysis to be even more complicated since whether a node should be moved from one level to another depends on its degree in one set, but the cost of moving such node depends on its degree in other sets as well. The analysis fortunately goes through (intuitively because all sets are sampled from the same graph and so their degree distributions are close enough). We explain our algorithm and how to analysis it in details in Section 4.

Notation. For any graph $G = (V, E)$, let $\mathcal{N}_v = \{u \in V : (u, v) \in E\}$ and $D_v = |\mathcal{N}_v|$ respectively denote the set of neighbors and the degree of a node $v \in V$. Let $G(S)$ denote the subgraph of G induced by the nodes in $S \subseteq V$. Given any two subsets $S \subseteq V, E' \subseteq E$, define $\mathcal{N}_u(S, E') = \{v \in \mathcal{N}_u \cap S : (u, v) \in E'\}$ and $D_u(S, E') = |\mathcal{N}_u(S, E')|$. To ease notation, we write $\mathcal{N}_u(S)$ and $D_u(S)$ instead of $\mathcal{N}_u(S, E)$ and $D_u(S, E)$. For a nonempty subset $S \subseteq V$, its density and average degree are defined as $\rho(S) = |E(S)|/|S|$ and $\delta(S) = \sum_{v \in S} D_v(S)/|S|$ respectively. Note that $\delta(S) = 2 \cdot \rho(S)$.

2. (α, D, L)-DECOMPOSITION

Our (α, d, L)-decomposition is formally defined as follows.

DEFINITION 2.1. *Fix any $\alpha \geq 1$, $d \geq 0$, and any positive integer L. Consider a family of subsets $Z_1 \supseteq \cdots \supseteq Z_L$. The tuple $(Z_1, \ldots, Z_L)$ is an (α, d, L)-decomposition of the input graph $G = (V, E)$ iff $Z_1 = V$ and, for every $i \in [L-1]$, we have $Z_{i+1} \supseteq \{v \in Z_i : D_v(Z_i) > \alpha d\}$ and $Z_{i+1} \cap \{v \in Z_i : D_v(Z_i) < d\} = \emptyset$.*

Given an (α, d, L)-decomposition $(Z_1, \ldots, Z_L)$, we define $V_i = Z_i \setminus Z_{i+1}$ for all $i \in [L-1]$, and $V_i = Z_i$ for $i = L$. We say that the nodes in V_i constitute the i^{th} level of this decomposition. We also denote the level of a node $v \in V$ by $\ell(v)$. Thus, we have $\ell(v) = i$ whenever $v \in V_i$. The following theorem and its immediate corollary will play the main role in the rest of the paper. Roughly speaking, they state that we can use the (α, d, L)-decomposition to $2\alpha(1+\epsilon)^2$-approximate the densest subgraph by setting $L = O(\log n/\epsilon)$ and trying different values of d in powers of $(1+\epsilon)$.

THEOREM 2.2. *Fix any $\alpha \geq 1$, $d \geq 0$, $\epsilon \in (0,1)$, $L \leftarrow 2 + \lceil \log_{(1+\epsilon)} n \rceil$. Let $d^* \leftarrow \max_{S \subseteq V} \rho(S)$ be the maximum density of any subgraph in $G = (V, E)$, and let $(Z_1, \ldots, Z_L)$ be an (α, d, L)-decomposition of $G = (V, E)$. We have*

- *(1) If $d > 2(1+\epsilon)d^*$, then $Z_L = \emptyset$.*
- *(2) Else if $d < d^*/\alpha$, then $Z_L \neq \emptyset$ and there is an index $j \in \{1, \ldots, L-1\}$ such that $\rho(Z_j) \geq d/(2(1+\epsilon))$.*

COROLLARY 2.3. *Fix α, ϵ, L, d^* as in Theorem 2.2. Let $\pi, \sigma > 0$ be any two numbers satisfying $\alpha \cdot \pi < d^* < \sigma/(2(1+\epsilon))$. Discretize the range $[\pi, \sigma]$ into powers of $(1+\epsilon)$, by defining $d_k \leftarrow (1+\epsilon)^{k-1} \cdot \pi$ for every $k \in [K]$, where K is any integer strictly greater than $\lceil \log_{(1+\epsilon)} (\sigma/\pi) \rceil$. For every $k \in [K]$, construct an (α, d_k, L)-decomposition $(Z_1(k), \ldots, Z_L(k))$ of $G = (V, E)$. Let $k' \leftarrow \max\{k \in [K] : Z_L(k) \neq \emptyset\}$. Then we have the following guarantees:*

- *$d^*/(\alpha(1+\epsilon)) \leq d_{k'} \leq 2(1+\epsilon) \cdot d^*$.*
- *There exists an index $j' \in \{1, \ldots, L-1\}$ such that $\rho(Z_{j'}(k')) \geq d_{k'}/(2(1+\epsilon))$.*

We will use the above corollary as follows. Since $K = O(\log_{1+\epsilon} n)$, it is not hard to maintain k' and the set of nodes $Z_{j'}(k')$. The corollary guarantees that the density of the set of nodes $Z_{j'}(k')$ is $(2\alpha(1+\epsilon)^2)$-approximation to d^*.

The next lemma relates the density to the minimum degree. Its proof can be found in the full version.

LEMMA 2.4. *Let $S^* \subseteq V$ be a subset of nodes with maximum density, i.e., $\rho(S^*) \geq \rho(S)$ for all $S \subseteq V$. Then $D_v(S^*) \geq \rho(S^*)$ for all $v \in S^*$. Thus, the degree of each node in $G(S^*)$ is at least the density of S^*.*

PROOF OF THEOREM 2.2. (1) Suppose that $d > 2(1+\epsilon)d^*$. Consider any level $i \in [L-1]$, and note that $\delta(Z_i) = 2 \cdot \rho(Z_i) \leq 2 \cdot \max_{S \subseteq V} \rho(S) = 2d^* < d/(1+\epsilon)$. It follows that the number of nodes v in $G(Z_i)$ with degree $D_v(Z_i) \geq d$ is less than $|Z_i|/(1+\epsilon)$, as otherwise $\delta(Z_i) \geq d/(1+\epsilon)$. Let us define the set $C_i = \{v \in Z_i : D_v(Z_i) < d\}$. We have $|Z_i \setminus C_i| \leq |Z_i|/(1+\epsilon)$. Now, from Definition 2.1 we have $Z_{i+1} \cap C_i = \emptyset$, which, in turn, implies that $|Z_{i+1}| \leq |Z_i \setminus C_i| \leq |Z_i|/(1+\epsilon)$. Thus, for all $i \in [L-1]$, we have $|Z_{i+1}| \leq |Z_i|/(1+\epsilon)$. Multiplying all these inequalities, for $i = 1$ to $L-1$, we conclude that $|Z_L| \leq |Z_1|/(1+\epsilon)^{L-1}$. Since $|Z_1| = |V| = n$ and $L = 2 + \lceil \log_{(1+\epsilon)} n \rceil$, we get $|Z_L| \leq n/(1+\epsilon)^{(1+\log_{(1+\epsilon)} n)} < 1$. This can happen only if $Z_L = \emptyset$.

(2) Suppose that $d < d^*/\alpha$, and let $S^* \subseteq V$ be a subset of nodes with highest density, i.e., $\rho(S^*) = d^*$. We will show that $S^* \subseteq Z_i$ for all $i \in \{1, \ldots, L\}$. This will imply that $Z_L \neq \emptyset$. Clearly, we have $S^* \subseteq V = Z_1$. By induction hypothesis, assume that $S^* \subseteq Z_i$ for some $i \in [L-1]$. We show that $S^* \subseteq Z_{i+1}$. By Lemma 2.4, for every node $v \in S^*$, we have $D_v(Z_i) \geq D_v(S^*) \geq \rho(S^*) = d^* > \alpha d$. Hence, from Definition 2.1, we get $v \in Z_{i+1}$ for all $v \in S^*$. This implies that $S^* \subseteq Z_{i+1}$.

Next, we will show that if $d < d^*/\alpha$, then there is an index $j \in \{1, \ldots, L-1\}$ such that $\rho(Z_j) \geq d/(2(1+\epsilon))$. For the sake of contradiction, suppose that this is not the case. Then we have $d < d^*/\alpha$ and $\delta(Z_i) = 2 \cdot \rho(Z_i) < d/(1+\epsilon)$ for every $i \in \{1, \ldots, L-1\}$. Then, applying an argument similar to case (1), we conclude that $|Z_{i+1}| \leq |Z_i|/(1+\epsilon)$ for every $i \in \{1, \ldots, L-1\}$, which implies that $Z_L = \emptyset$. Thus, we arrive at a contradiction. □

3. WARMUP: A SINGLE PASS STREAMING ALGORITHM

In this section, we present a single-pass streaming algorithm for maintaining a $(2+\epsilon)$-approximate solution to the densest subgraph problem. The algorithm handles a dynamic (turnstile) stream of edge insertions/deletions in $\tilde{O}(n)$ space. In particular, we do not worry about the update time of our algorithm. Our main result in this section is summarized in Theorem 3.1.

THEOREM 3.1. *We can process a dynamic stream of updates in the graph G in $\tilde{O}(n)$ space, and with high probability return a $(2 + O(\epsilon))$-approximation of $d^* = \max_{S \subseteq V} \rho(S)$ at the end of the stream.*

Throughout this section, we fix a small constant $\epsilon \in (0, 1/2)$ and a sufficiently large constant $c > 1$. Moreover, we set $\alpha \leftarrow (1+\epsilon)/(1-\epsilon)$, $L \leftarrow 2 + \lceil \log_{(1+\epsilon)} n \rceil$. The main technical lemma is below and states that we can construct a (α, d, L)-decomposition by sampling $\tilde{O}(n)$ edges.

LEMMA 3.2. *Fix an integer $d > 0$, and let S be a collection of $(cm(L-1)\log n)/d$ mutually independent random samples (each consisting of one edge) from the edge-set E of the input graph $G = (V, E)$. With high probability we can construct from S an (α, d, L)-decomposition $(Z_1, \ldots, Z_L)$ of G, using only $\tilde{O}((n + m/d))$ bits of space.*

PROOF. We partition the samples in S evenly among $(L-1)$ groups $\{S_i\}, i \in [L-1]$. Thus, each S_i is a collection of $(cm \log n)/d$ mutually independent random samples from the edge-set E, and, furthermore, the collections $\{S_i\}, i \in [L-1]$, themselves are mutually independent. Our algorithm works as follows.

- Set $Z_1 \leftarrow V$.
- FOR $i = 1$ to $(L-1)$: Set $Z_{i+1} \leftarrow \{v \in Z_i : D_v(Z_i, S_i) \geq (1-\epsilon)\alpha c \log n\}$.

To analyze the correctness of the algorithm, define the (random) sets $A_i = \{v \in Z_i : D_v(Z_i, E) > \alpha d\}$ and $B_i = \{v \in Z_i : D_v(Z_i, E) < d\}$ for all $i \in [L-1]$. Note that for all $i \in [L-1]$, the random sets Z_i, A_i, B_i are completely determined by the outcomes of the samples in $\{S_j\}, j < i$. In particular, the samples in S_i are chosen independently of the sets Z_i, A_i, B_i. Let $\mathcal{E}_i$ be the event that (a) $Z_{i+1} \supseteq A_i$ and (b) $Z_{i+1} \cap B_i = \emptyset$. By Definition 2.1, the output $(Z_1, \ldots, Z_L)$ is a valid (α, d, L)-decomposition of G iff the event $\bigcap_{i=1}^{L-1} \mathcal{E}_i$ occurs. Consider any $i \in [L-1]$. Below, we show that the event $\mathcal{E}_i$ occurs with high probability. The lemma follows by taking a union bound over all $i \in [L-1]$.

Fix any instantiation of the random set Z_i. Condition on this event, and note that this event completely determines the sets A_i, B_i. Consider any node $v \in A_i$. Let $X_{v,i}(j) \in \{0,1\}$ be an indicator random variable for the event that the j^{th} sample in S_i is of the form (u,v), with $u \in \mathcal{N}_v(Z_i)$. Note that the random variables $\{X_{v,i}(j)\}, j$, are mutually independent. Furthermore, we have $E[X_{v,i}(j)|Z_i] = D_v(Z_i)/m > \alpha d/m$ for all j. Since there are $cm \log n/d$ such samples in S_i, by linearity of expectation we get: $E[D_v(Z_i, S_i)|Z_i] = \sum_j E[X_{v,i}(j)|Z_i] > (cm \log n/d) \cdot (\alpha d/m) = \alpha c \log n$. The node v is included in Z_{i+1} iff $D_v(Z_i, S_i) \geq (1-\epsilon)\alpha c \log n$, and this event, in turn, occurs with high probability (by Chernoff bound). Taking a union bound over all nodes $v \in A_i$, we conclude that $\Pr[Z_{i+1} \supseteq A_i \,|\, Z_i] \geq 1 - 1/(\text{poly } n)$. Using a similar line of reasoning, we get that $\Pr[Z_{i+1} \cap B_i = \emptyset \,|\, Z_i] \geq 1 - 1/(\text{poly } n)$. Invoking a union bound over these two events, we get $\Pr[\mathcal{E}_i \,|\, Z_i] \geq 1 - 1/(\text{poly } n)$. Since this holds for all possible instantiations of Z_i, the event $\mathcal{E}_i$ itself occurs with high probability.

The space requirement of the algorithm, ignoring poly log factors, is proportional to the number of samples in S (which is $cm(L-1)\log n/d$) plus the number of nodes in V (which is n). Since c is a constant and since $L = O(\text{poly} \log n)$, we derive that the total space requirement is $O((n + m/d)\ \text{poly} \log n)$. □

Now, to turn Lemma 3.2 into a streaming algorithm, we simply have to invoke Lemma 3.3 which follows from a well-known result about ℓ_0-sampling in the streaming model [24], and a simple observation (yet very important) in Lemma 3.4.

LEMMA 3.3 (ℓ_0-SAMPLER [24]). *We can process a dynamic stream of $O(\text{poly } n)$ updates in the graph $G = (V, E)$ in $O(\text{poly} \log n)$ space, and with high probability, at each step we can maintain a simple random sample from the set E. The algorithm takes $O(\text{poly} \log n)$ time to handle each update in the stream.*

LEMMA 3.4. *Let $d^* = \max_{S \subseteq V} \rho(S)$ be the maximum density of any subgraph in G. Then $m/n \leq d^* < n$.*

PROOF OF THEOREM 3.1. Since m is the number of edges in the graph G at the end of the entire stream of updates, we do not know the value of m in advance. Thus, before processing the first update in the stream, we make $O(\text{poly}(\log n, 1/\epsilon))$ many "guesses" for the value of m. Let m' be the "correct" guess (within a multiplicative factor of $1+\epsilon$) for the value of m. Define $\pi \leftarrow m'/(2\alpha n)$ and $\sigma \leftarrow 2(1+\epsilon)n$. Since $\epsilon \in (0, 1/2)$, by Lemma 3.4 we have $\alpha \cdot \pi < d^* < \sigma/(2(1+\epsilon))$. Thus, we can discretize the range $[\pi, \sigma]$ in powers of $(1+\epsilon)$ by defining the values $\{d_k\}, k \in [K]$, as per Corollary 2.3. Accordingly, to return a $2\alpha(1+\epsilon)^2 = (2 + O(\epsilon))$-approximation of optimal density, all we need to do is to construct an (α, d_k, L)-decomposition of the graph $G = (V, E)$ at the end of the stream, for every $k \in [K]$. Since $K = O(\log_{(1+\epsilon)}(\sigma/\pi)) = O(\text{poly} \log n)$, Theorem 3.1 follows from Claim 3.5 below. For each of our initial guesses for the value of m, we run the above procedure (in parallel) while going through the stream of updates. At the end of the stream, we simply reject the outputs given by the wrong guesses and take the output given by the correct one.

CLAIM 3.5. *Fix any $k \in [K]$. We can process a dynamic stream of updates in the graph G in $O(n\ \text{poly} \log n)$ space, and with high probability return an (α, d_k, L)-decomposition of G at the end of the stream.*

Now we prove Claim 3.5. Define $\lambda_k \leftarrow cm(L-1)\log n/d_k$. Since $d_k \geq \pi = m/(2\alpha n)$, we have $\lambda_k = O(n \text{ poly} \log n)$. While going through the dynamic stream of updates in G, we simultaneously run λ_k mutually independent copies of the ℓ_0-sampler as specified in Lemma 3.3. Thus, with high probability, we get λ_k mutually independent simple random samples from the edge-set E at the end of the stream. Next, we use these random samples to construct an (α, d_k, L)-decomposition of G, with high probability, as per Lemma 3.2.

By Lemma 3.3, each ℓ_0-sampler requires $O(\text{poly} \log n)$ bits of space, and there are λ_k many of them. Furthermore, the algorithm in Lemma 3.2 requires $O((n + m/d_k)\ \text{poly} \log n)$ bits of space. Thus, the total space requirement of our algorithm is $O((\lambda_k + n + m/d_k)\ \text{poly} \log n) = O(n\ \text{poly} \log n)$ bits. □

4. A SINGLE PASS DYNAMIC STREAMING ALGORITHM

We devote this section to the proof of our main result (Theorem 1.1). Throughout this section, fix $\alpha = 2 + \Theta(\epsilon)$, $L \leftarrow 2 + \lceil \log_{(1+\epsilon)} n \rceil$, and let $c \gg \lambda$ be a sufficiently large constant. We call the input graph "sparse" whenever it has less than $4\alpha c^2 n \log^2 n$ edges, and "dense" otherwise. We simultaneously run two algorithms while processing the stream of updates – the first (resp. second) one outputs a correct value whenever the graph is sparse (resp. dense). It is the algorithm for dense graphs that captures the technical difficulty of the problem. To focus on this case (due to space constraints), we assume that the first $4\alpha c^2 n \log^2 n$ updates in the dynamic stream consist of only edge-insertions, so that the graph $G^{(t)}$ becomes dense at $t = 4\alpha c^2 n \log^2 n$. Next, we assume that the graph $G^{(t)}$ remains dense at each $t \geq 4\alpha c^2 n \log^2 n$. We focus on maintaining the value of $\text{OUTPUT}^{(t)}$ during the latter phase. For a full proof of Theorem 1.1 that does not require any of these simplifying assumptions, see the full version [7].

ASSUMPTION 4.1. *Define $T' \leftarrow \lceil 4\alpha c^2 n \log^2 n \rceil$. We have $m^{(t)} \geq 4\alpha c^2 n \log^2 n$ for all $t \in [T', T]$.*

Consider any $t \in [T', T]$. Define $\pi^{(t)} = m^{(t)}/(2\alpha n)$ and $\sigma = 2(1+\epsilon)n$. It follows that $\alpha \cdot \pi^{(t)} < \text{OPT}^{(t)} < \sigma/(2(1+\epsilon))$. Discretize the range $[\pi^{(t)}, \sigma]$ in powers of $(1+\epsilon)$, by defining $d_k^{(t)} \leftarrow (1+\epsilon)^{k-1} \cdot \pi^{(t)}$ for all $k \in [K]$, where $K \leftarrow 1 + \lceil \log_{(1+\epsilon)}(\sigma \cdot (2\alpha n)) \rceil$. Note that for all $t \in [T', T]$ we have $K > \lceil \log_{(1+\epsilon)}(\sigma/\pi^{(t)}) \rceil$. Also note that $K = O(\text{poly} \log n)$. By Corollary 2.3, the algorithm only has to maintain an $(\alpha, d_k^{(t)}, L)$-decomposition for each $k \in [K]$. Specifically, Theorem 1.1 follows from Theorem 4.2.

THEOREM 4.2. *Let us fix any $k \in [K]$. There is an algorithm that processes the first T updates in the dynamic stream using $O(n)$ space, and under Assumption 4.1, it gives the following guarantees with high probability: At each $t \in [T', T]$, the algorithm maintains an $(\alpha, d_k^{(t)}, L)$-decomposition $(Z_1^{(t)}, \ldots, Z_L^{(t)})$ of $G^{(t)}$. Further, the total amount of computation performed is $O(T\ \text{poly} \log n)$.*

As we mentioned earlier in Sections 1 and 2, our algorithm can output an approximate densest subgraph by maintaining

the density at each level of the (α, d, L) decomposition and simply keeping track of the level that gives us maximum density.

4.1 Proof of Theorem 4.2

Notation. Define $s_k^{(t)} = cm^{(t)} \log n / d_k^{(t)}$ for all $t \in [T', T]$. Plugging in the value of $d_k^{(t)}$, we get $s_k^{(t)} = 2\alpha cn \log n/(1+\epsilon)^{k-1}$. Since $s_k^{(t)}$ does not depend on t, we omit the superscript and refer to it as s_k instead.

Overview of our approach. As a first step, we want to show that for each $i \in [L-1]$, we can maintain a random set of s_k edges $S_i^{(t)} \subseteq E^{(t)}$ such that $\Pr[e \in S_i^{(t)}] = s_k/m^{(t)}$ for all $e \in E^{(t)}$. This has the following implication: Fix any subset of nodes $U \subseteq V$. If a node $u \in U$ has $D_u(U, E^{(t)}) > \alpha d_k^{(t)}$, then in expectation we have $D_u(U, S_i^{(t)}) > \alpha c \log n$. Since this expectation is large enough, a suitable Chernoff bound implies that $D_u(U, S_i^{(t)}) > (1-\epsilon)\alpha c \log n$ with high probability. Accordingly, we can use the random sets $\{S_i^{(t)}\}$, $i \in [L-1]$, to construct an $(\alpha, d_k^{(t)}, L)$-decomposition of $G^{(t)}$ as follows. We set $Z_1^{(t)} = V$, and for each $i \in [L-1]$, we iteratively construct the subset $Z_{i+1}^{(t)}$ by taking the nodes $u \in Z_i^{(t)}$ with $D_u(Z_i^{(t)}, S_i^{(t)}) > (1-\epsilon)\alpha c \log n$. Here, we crucially need the property that the random set $S_i^{(t)}$ is chosen independently of the contents of $Z_i^{(t)}$. Note that $Z_i^{(t)}$ is actually determined by the contents of the sets $\{S_j^{(t)}\}, j < i$. Since $s_k = \tilde{O}(n)$, each of these random sets $S_i^{(t)}$ consists of $\tilde{O}(n)$ many edges. While following up on this high level approach, we need to address two major issues, as described below.

Fix some $i \in [L-1]$. A naive way of maintaining the set $S_i^{(t)}$ would be to invoke a well known result on ℓ_0-sampling on dynamic streams (see Lemma 3.3). This allows us to maintain a uniformly random sample from $E^{(t)}$ in $\tilde{O}(1)$ update time. So we might be tempted to run s_k mutually independent copies of such an ℓ_0-Sampler on the edge-set $E^{(t)}$ to generate a random set of size s_k. The problem is that when an edge insertion/deletion occurs in the input graph, we have to probe each of these ℓ_0-Samplers, leading to an overall update time of $O(s_k \operatorname{poly}\log n)$, which can be as large as $\tilde{\Theta}(n)$ when k is small (say for $k = 1$). In Lemma 4.3, we address this issue by showing how to maintain the set $S_i^{(t)}$ in $\tilde{O}(1)$ worst case update time and $\tilde{O}(n)$ space.

The remaining challenge is to maintain the decomposition $(Z_1^{(t)}, \ldots, Z_L^{(t)})$ dynamically as the random sets $\{S_i^{(t)}\}, i \in [L-1]$, change with t. Again, a naive implementation – building the decomposition from scratch at each t – would require $\Theta(n)$ update time. In Section 4.1.1, we give a procedure that builds a new decomposition at any given $t \in [T', T]$, based on the old decomposition at $(t-1)$ and the new random sets $\{S_i^{(t)}\}, i \in [L-1]$. In Section 4.1.2, we present the data structures for implementing this procedure and analyze the space complexity. In Section 4.1.3, we bound the amortized update time using a fine-tuned potential function. Theorem 4.2 follows from Lemmata 4.5, 4.6, 4.10 and Claim 4.9.

Lemma 4.3. *We can process the first T updates in a dynamic stream using $\tilde{O}(n)$ space and maintain a random subset of edges $S_i^{(t)} \subseteq E^{(t)}, |S_i^{(t)}| = s_k$, at each $t \in [T', T]$. Let $X_{e,i}^{(t)}$ denote an indicator variable for the event $e \in S_i^{(t)}$. The following guarantee holds w.h.p.:*

- *At each $t \in [T', T]$, we have that $\Pr[X_{e,i}^{(t)} = 1] \in \left[(1 \pm \epsilon) c \log n / d_k^{(t)}\right]$ for all $e \in E^{(t)}$. The variables $\left\{X_{e,i}^{(t)}\right\}, e \in E^{(t)}$, are negatively associated.*
- *Each update in the dynamic stream is handled in $\tilde{O}(1)$ time and leads to at most two changes in S_i.*

Proof. (Sketch) Let E^* denote the set of all possible ordered pairs of nodes in V. Thus, $E^* \supseteq E^{(t)}$ at each $t \in [1, T]$, and furthermore, we have $|E^*| = O(n^2)$. Using a well known result from the hashing literature [34], we construct a $(2cs_k \log n)$-wise independent uniform hash function $h : E^* \to [s_k]$ in $\tilde{O}(n)$ space. This hash function partitions the edge-set $E^{(t)}$ into s_k mutually disjoint buckets $\{Q_j^{(t)}\}, j \in [s_k]$, where the bucket $Q_j^{(t)}$ consists of those edges $e \in E^{(t)}$ with $h(e) = j$. For each $j \in [s_k]$, we run an independent copy of ℓ_0-Sampler, as per Lemma 3.3, that maintains a uniformly random sample from $Q_j^{(t)}$. The set $S_i^{(t)}$ consists of the collection of outputs of all these ℓ_0-Samplers. Note that (a) for each $e \in E^*$, the hash value $h(e)$ can be evaluated in constant time [34], (b) an edge insertion/deletion affects exactly one of the buckets, and (c) the ℓ_0-Sampler of the affected bucket can be updated in $\tilde{O}(1)$ time. Thus, we infer that this procedure handles an edge insertion/deletion in the input graph in $\tilde{O}(1)$ time, and furthermore, since $s_k = \tilde{O}(n)$, the procedure can be implemented in $\tilde{O}(n)$ space.

Fix any time-step $t \in [T', T]$ (see Assumption 4.1). Since $m^{(t)} \geq 2cs_k \log n$, we can partition (purely as a thought experiment) the edges in $E^{(t)}$ into at most polynomially many groups $\left\{H_{j'}^{(t)}\right\}$, in such a way that the size of each group lies between $cs_k \log n$ and $2cs_k \log n$. Thus, for any $j \in [s_k]$ and any j', we have $|H_{j'}^{(t)} \cap Q_j^{(t)}| \in [c \log n, 2c \log n]$ in expectation. Since the hash function h is $(2cs_k \log n)$-wise independent, by applying a Chernoff bound we infer that with high probability, the value $|H_{j'}^{(t)} \cap Q_j^{(t)}|$ is very close to its expectation. Applying the union bound over all j, j', we infer that with high probability, the sizes of all the sets $\left\{H_{j'}^{(t)} \cap Q_j^{(t)}\right\}$ are very close to their expected values – let us call this event $\mathcal{R}^{(t)}$. Since $E[|Q_j^{(t)}|] = m^{(t)}/s_k$ and $|Q_j^{(t)}| = \sum_{j'} |Q_j^{(t)} \cap H_{j'}^{(t)}|$, under the event $\mathcal{R}^{(t)}$, we have that $|Q_j^{(t)}|$ is very close to $m^{(t)}/s_k$ for all $j \in [s_k]$. Under the same event $\mathcal{R}^{(t)}$, due to the ℓ_0-Samplers, the probability that a given edge $e \in E^{(t)}$ becomes part of $S_i^{(t)}$ is very close to $1/|Q_j^{(t)}| \approx s_k/m^{(t)} = c \log n / d_k^{(t)}$.

Finally, the property of negative association follows from the observations that (a) if two edges are hashed to different buckets, then they are included in $S_i^{(t)}$ in a mutually independent manner, and (b) if they are hashed to the same bucket, then they are never simultaneously included in $S_i^{(t)}$. □

4.1.1 Maintaining an $(\alpha, d_k^{(t)}, L)$-decomposition using the random sets $S_i^{(t)}, i \in [L-1]$

While processing the stream of updates, we run an independent copy of the algorithm in Lemma 4.3 for each $i \in [L-1]$. Thus, we assume that we have access to the random sets $S_i^{(t)}, i \in [L-1]$, at each $t \in [T', T]$. In this section, we present an algorithm that maintains a decomposition $(Z_1^{(t)}, \ldots, Z_L^{(t)})$ at each time-step $t \in [T', T]$ as long as the graph is dense (see Assumption 4.1), using the random sets $S_i^{(t)}, i \in [L-1]$. Specifically, we handle the t^{th} update in the dynamic stream as per the procedure in Algorithm 1. The procedure outputs the new decomposition $(Z_1^{(t)}, \ldots, Z_L^{(t)})$ starting from the old decomposition $(Z_1^{(t-1)}, \ldots, Z_L^{(t-1)})$ and the new samples $\left\{S_i^{(t)}\right\}, i \in [L-1]$.

Algorithm 1 RECOVER-SAMPLE(t).

1: Set $Z_1^{(t)} \leftarrow V$.
2: **for** $i = 1$ to L **do**
3: Set $Y_i \leftarrow Z_i^{(t-1)}$.
4: **end for**
5: **for** $i = 1$ to $(L-1)$ **do**
6: Let $A_i^{(t)}$ be the set of nodes $y \in Z_i^{(t)}$ having $D_y(Z_i^{(t)}, S_i^{(t)}) > (1-\epsilon)^2 \alpha c \log n$.
7: Let $B_i^{(t)}$ be the set of nodes $y \in Z_i^{(t)}$ having $D_y(Z_i^{(t)}, S_i^{(t)}) < (1+\epsilon)^2 c \log n$.
8: Set $Y_{i+1} \leftarrow Y_{i+1} \cup A_i^{(t)}$.
9: **for** $j = i+1$ to $(L-1)$ **do**
10: Set $Y_j \leftarrow Y_j \setminus B_i^{(t)}$.
11: **end for**
12: Set $Z_{i+1}^{(t)} \leftarrow Y_{i+1}$.
13: **end for**

We have have the following observation.

LEMMA 4.4. *Fix a $t \in [T', T]$ and an $i \in [L-1]$. (1) The set $Z_i^{(t)}$ is completely determined by the contents of the sets $\left\{S_j^{(t)}\right\}, j < i$. (2) The sets $\left\{S_j^{(t)}\right\}, j \geq i$, are chosen independently of the contents of the set $Z_i^{(t)}$.*

LEMMA 4.5. *With high probability, at each $t \in [T', T]$ the tuple $(Z_1^{(t)} \ldots Z_L^{(t)})$ is an $(\alpha, d_k^{(t)}, L)$-decomposition of $G^{(t)}$.*

PROOF. (sketch) For $t \in [T', T], i \in [L-1]$, let $\mathcal{E}_i^{(t)}$ denote the event that (a) $Z_{i+1}^{(t)} \supseteq \{v \in Z_i^{(t)} : D_v(Z_i^{(t)}, E^{(t)}) > \alpha d_k^{(t)}\}$ and (b) $Z_{i+1}^{(t)} \cap \{v \in Z_i^{(t)} : D_v(Z_i^{(t)}, E^{(t)}) < d_k^{(t)}\} = \emptyset$. By Definition 2.1, the tuple $(Z_1^{(t)} \ldots Z_L^{(t)})$ is an $(\alpha, d_k^{(t)}, L)$-decomposition of $G^{(t)}$ iff the event $\mathcal{E}_i^{(t)}$ holds for all $i \in [L-1]$. Below, we show that $\Pr[\mathcal{E}_i^{(t)}] \geq 1 - 1/(\text{poly } n)$ for any given $i \in [L-1]$ and $t \in [T', T]$. The lemma follows by taking a union bound over all i, t.

Fix any instance of the random set $Z_i^{(t)}$ and condition on this event. Consider any node $v \in Z_i^{(t)}$ with $D_v(Z_i^{(t)}, E^{(t)}) > \alpha d_k^{(t)}$. By Lemma 4.3, each edge $e \in E^{(t)}$ appears in $S_i^{(t)}$ with probability $(1 \pm \epsilon) c \log n / d_k^{(t)}$ and these events are negatively associated. By linearity of expectation, we have $E[D_v(Z_i^{(t)}, S_i^{(t)})] \geq (1-\epsilon)\alpha c \log n$. Since the random set $S_i^{(t)}$ is chosen independently of the contents of $Z_i^{(t)}$ (see Lemma 4.4), we can apply a Chernoff bound on this expectation and derive that $\Pr[v \notin Z_{i+1}^{(t)} \mid Z_i^{(t)}] = \Pr[D_v(Z_i^{(t)}, S_i^{(t)}) \leq (1-\epsilon)^2 \alpha c \log n \mid Z_i^{(t)}] \leq 1/(\text{poly } n)$. Next, consider any node $u \in Z_i^{(t)}$ with $D_u(Z_i^{(t)}, E^{(t)}) < d_k^{(t)}$. Using a similar argument, we get $\Pr[u \in Z_{i+1}^{(t)} \mid Z_i^{(t)}] = \Pr[D_u(Z_i^{(t)}, E^{(t)}) \geq (1+\epsilon)^2 c \log n \mid Z_i^{(t)}] \leq 1/(\text{poly } n)$. Taking a union bound over all possible nodes, we infer that $\Pr[\mathcal{E}_i^{(t)} \mid Z_i^{(t)}] \geq 1 - 1/(\text{poly } n)$.

Since the guarantee $\Pr[\mathcal{E}_i^{(t)} \mid Z_i^{(t)}] \geq 1 - 1/(\text{poly } n)$ holds for every possible instance of $Z_i^{(t)}$, we get $\Pr[\mathcal{E}_i^{(t)}] \geq 1 - 1/(\text{poly } n)$. □

4.1.2 Data structures for the procedure in Algorithm 1 and bounding its amortized update time

Recall the notations defined immediately after Definition 2.1.

- Consider any node $v \in V$ and any $i \in \{1, \ldots, L-1\}$. We maintain the doubly linked lists $\{\text{FRIENDS}_i[v, j]\}, 1 \leq j \leq L-1$ as defined below. These lists depend on the neighborhood of v induced by the edge-set S_i.
  - If $i \leq \ell(v)$, then we have:
    - $\text{FRIENDS}_i[v, j]$ is empty for all $j > i$.
    - $\text{FRIENDS}_i[v, j] = \mathcal{N}_v(Z_j, S_i)$ for $j = i$.
    - $\text{FRIENDS}_i[v, j] = \mathcal{N}_v(V_j, S_i)$ for all $j < i$.
  - Else if $i > \ell(v)$, then we have:
    - $\text{FRIENDS}_i[v, j]$ is empty for all $j > \ell(v)$.
    - $\text{FRIENDS}_i[v, j] = \mathcal{N}_v(Z_j, S_i)$ for $j = \ell(v)$.
    - $\text{FRIENDS}_i[v, j] = \mathcal{N}_v(V_j, S_i)$ for all $j < \ell(v)$.

For every node $v \in V$, we maintain a counter $\text{DEGREE}_i[v]$. If $\ell(v) \geq i$, then this counter equals the number of nodes in $\text{FRIENDS}_i[v, i]$. Else if $\ell(v) < i$, then this counter equals zero. Further, we maintain a doubly linked list $\text{DIRTY-NODES}[i]$. This list consists of all the nodes $v \in V$ having either $\left\{\text{DEGREE}_i[v] > (1-\epsilon)^2 \alpha c \log n \text{ and } \ell(v) = i\right\}$ or $\left\{\text{DEGREE}_i[v] < (1+\epsilon)^2 c \log n \text{ and } \ell(v) > i\right\}$.

Implementing the procedure in Algorithm 1. Fix any $t \in [T', T]$, and consider the i^{th} iteration of the main FOR loop (Steps 05-13) in Algorithm 1. The purpose of this iteration is to construct the set $Z_{i+1}^{(t)}$, based on the sets $Z_i^{(t)}$ and $S_i^{(t)}$. Below, we state an alternate way of visualizing this iteration.

We scan through the list of nodes u with $\ell(u) = i$ and $D_u(Z_i^{(t)}, S_i^{(t)}) > (1-\epsilon)^2 \alpha c \log n$. While considering each such node u, we increment its level from i to $(i+1)$. This takes care of the Steps (06) and (08). Next, we scan through the list of nodes v with $\ell(v) > i$ and $D_v(Z_i^{(t)}, S_i^{(t)}) < (1+\epsilon)^2 c \log n$. While considering any such node v at level $\ell(v) = j_v > i$ (say), we decrement its level from j_v to i. This takes care of the Steps (07), (09), (10) and (11).

Note that the nodes undergoing a level-change in the preceding paragraph are precisely the ones that appear in the list $\text{DIRTY-NODES}[i]$ just before the i^{th} iteration of the main FOR loop. Thus, we can implement a single iteration of the For loop (Steps 05-13) as follows: Scan through the nodes y in $\text{DIRTY-NODES}[i]$ one after another. While considering any such node y, change its level as per Algorithm 1, and then update the relevant data structures to reflect this change.

LEMMA 4.6. *The procedure in Algorithm 1 can be implemented in $\tilde{O}(n)$ space.*

PROOF. (sketch) The amount of space needed is dominated by the number of edges in $\left\{S_i^{(t)}\right\}, i \in [L-1]$. Since $|S_i^{(t)}| \leq s_k$ for each $i \in [L-1]$, the space complexity is $(L-1) \cdot s_k = \tilde{O}(n)$. $\square$

CLAIM 4.7. *Fix a $t \in [T', T]$ and consider the i^{th} iteration of the main* FOR *loop in Algorithm 1. Consider any two nodes $u, v \in Z_i^{(t)}$ such that (a) the level of u is increased from i to $(i+1)$ in Steps 08, 12 and (b) the level of v is decreased to i in Steps 09-12. Updating the relevant data structures require $\sum_{i'>i} O(1 + D_y(Z_i^{(t)}, S_{i'}^{(t)}))$ time, where $y = u$ (resp. v) in the former (resp. latter) case.*

PROOF. (sketch) Follows from the fact that we only need to update the lists $\text{FRIENDS}_{i'}[x, j]$ where $i' > i$, $x \in \{y\} \cup \mathcal{N}_y(Z_i^{(t)}, S_{i'}^{(t)})$ and $j \in \{1, \ldots, L-1\}$. $\square$

4.1.3 Bounding the amortized update time

Potential function. To determine the amortized update time we use a potential function $\mathcal{B}$ as defined in equation 4. Note that the potential $\mathcal{B}$ is uniquely determined by the assignment of the nodes $v \in V$ to the levels $[L]$ and by the content of the random sets $S_1, \ldots, S_{(L-1)}$. For all nodes $v \in V$, we define:

$$\Gamma_i(v) = \max(0, (1-\epsilon)^2 \alpha c \log n - D_v(Z_i, S_i)) \quad (1)$$

$$\Phi(v) = (L/\epsilon) \cdot \sum_{i=1}^{\ell(v)-1} \Gamma_i(v) \quad (2)$$

For all $u, v \in V$, let $f(u,v) = 1$ if $\ell(u) = \ell(v)$ and 0 otherwise. Also, let $r_{uv} = \min(\ell(u), \ell(v))$. For all $i \in [L-1]$, $(u,v) \in S_i$, we define:

$$\Psi_i(u,v) = \begin{cases} 0 & \text{if } r_{uv} \geq i; \\ 2 \cdot (i - r_{uv}) + f(u,v) & \text{otherwise.} \end{cases}, \quad (3)$$

$$\mathcal{B} = \sum_{v \in V} \Phi(v) + \sum_{i=1}^{(L-1)} \sum_{e \in S_i} \Psi_i(e) \quad (4)$$

Below, we show that an event $\mathcal{F}$ holds with high probability (Definition 4.8, Claim 4.9). Next, conditioned on this event, we show that our algorithm has $O(\text{poly} \log n)$ amortized update time (Lemma 4.10).

DEFINITION 4.8. *For all $i, i' \in [L-1]$, $i < i'$, let $\mathcal{F}_{i,i'}^{(t)}$ be the event that: $\left\{D_v(Z_i^{(t)}, S_{i'}^{(t)}) \geq \frac{(1-\epsilon)^4}{(1+\epsilon)^2} \cdot (\alpha c \log n)\right.$ for all $v \in A_i^{(t)}\left.\right\}$, and $\left\{D_v(Z_i^{(t)}, S_{i'}^{(t)}) \leq \frac{(1+\epsilon)^4}{(1-\epsilon)^2} \cdot c \log n\right.$ for all $v \in B_i^{(t)}\left.\right\}$. Define $\mathcal{F}^{(t)} = \bigcap_{i,i'} \mathcal{F}_{i,i'}^{(t)}$.*

CLAIM 4.9. *Define the event $\mathcal{F} = \bigcap_{t=T'}^{T} \mathcal{F}^{(t)}$. The event $\mathcal{F}$ holds with high probability.*

PROOF. (sketch) Fix any $1 \leq i < i' \leq L-1$, any $t \in [T', T]$, and condition on any instance of the random set $Z_i^{(t)}$. By Lemma 4.4, the random sets $S_i^{(t)}, S_{i'}^{(t)}$ are chosen independently of $Z_i^{(t)}$. Further, for all $v \in Z_i^{(t)}$, we have $E[D_v(Z_i^{(t)}, S_i^{(t)})] = E[D_v(Z_i^{(t)}, S_{i'}^{(t)})] = (c \log n / d_k^{(t)}) \cdot D_v(Z_i^{(t)}, E^{(t)})$, and by Lemma 4.3 we can apply a Chernoff bound on this expectation. Thus, applying union bounds over $\{i, i'\}$, we infer that w.h.p. the following condition holds: If $D_v(Z_i^{(t)}, E^{(t)})$ is sufficiently smaller (resp. larger) than $d_k^{(t)}$, then both $D_v(Z_i^{(t)}, S_i^{(t)})$ and $D_v(Z_i^{(t)}, S_{i'}^{(t)})$ are sufficiently smaller (resp. larger) than $c \log n$. The proof follows by deriving a variant of this claim and then applying union bounds over all i, i' and t. $\square$

LEMMA 4.10. *Condition on event $\mathcal{F}$. We have (a) $0 \leq \mathcal{B} = \tilde{O}(n)$ at each $t \in [T', T]$, (b) insertion/deletion of an edge in G (ignoring the call to Algorithm 1) changes the potential $\mathcal{B}$ by $\tilde{O}(1)$, and (c) for every constant amount of computation performed while implementing Algorithm 1, the potential $\mathcal{B}$ drops by $\Omega(1)$.*

Theorem 4.2 follows from Lemmata 4.5, 4.6, 4.10 and Claim 4.9. We now focus on proving Lemma 4.10.

Proof of part (a). Follows from three facts. (1) We have $0 \leq \Phi(v) \leq (L/\epsilon) \cdot L \cdot (1-\epsilon)^2 \alpha c \log n = O(\text{poly} \log n)$ for all $v \in V$. (2) We have $0 \leq \Psi_i(u,v) \leq 3L = O(\text{poly} \log n)$ for all $i \in [L-1], (u,v) \in S_i^{(t)}$. (3) We have $|S_i^{(t)}| \leq s_k = O(n \text{ poly} \log n)$ for all $i \in [L-1]$.

Proof of part (b). By Lemma 4.3, insertion/deletion of an edge in G leads to at most two insertions/deletions in the random set S_i, for all $i \in [L-1]$. As $L = O(\text{poly} \log n)$, it suffices to show that for every edge insertion/deletion in any given $S_i^{(t)}$, the potential $\mathcal{B}$ changes by at most $O(\text{poly} \log n)$ (ignoring call to Algorithm 1).

Towards this end, fix any $i \in [L-1]$, and suppose that a single edge (u,v) is inserted into (resp. deleted from) $S_i^{(t)}$. For each node $v \in V$, this changes the potential $\Phi(v)$ by at most $O(L/\epsilon)$. Additionally, the potential $\Psi_i(u,v) \in [0, 3L]$ is created (resp. destroyed). Summing over all the nodes $v \in V$, we infer that the absolute value of the change in the overall potential $\mathcal{B}$ is at most $O(3L + nL/\epsilon) = O(n \text{ poly} \log n)$.

Proof of part (c). Focus on a single iteration of the FOR loop in Algorithm 1. Consider two possible operations.

CASE 1: A NODE $v \in Z_i^{(t)}$ IS PROMOTED FROM LEVEL i TO LEVEL $(i+1)$ IN STEPS 08, 12 OF ALGORITHM 1.

This can happen only if $v \in A_i^{(t)}$. Let C denote the amount of computation performed during this event.

$$C = \sum_{i'=(i+1)}^{(L-1)} O\left(1 + D_v(Z_i^{(t)}, S_{i'}^{(t)})\right) \quad (5)$$

Let Δ be the net decrease in the overall potential $\mathcal{B}$ due to this event. We make the following observations.

1. Consider any $i' > i$. For each edge $(u,v) \in S_{i'}^{(t)}$ with $u \in Z_i^{(t)}$, the potential $\Psi_{i'}(u,v)$ decreases by at least one. For every other edge $e \in S_{i'}^{(t)}$, the potential $\Psi_{i'}(e)$ remains unchanged.

2. For each $i' \in [i]$ and each edge $e \in S_{i'}^{(t)}$, the potential $\Psi_{i'}(e)$ remains unchanged.

3. Since the node v is being promoted to level $(i+1)$, we have $D_v(Z_i^{(t)}, S_i^{(t)}) \geq (1-\epsilon)^2 \alpha c \log n$. Thus, the potential $\Phi(v)$ remains unchanged. For each node $u \neq v$, the potential $\Phi(u)$ can only decrease.

Taking into account all these observations, we infer the following inequality.

$$\Delta \geq \sum_{i'=(i+1)}^{(L-1)} D_v(Z_i^{(t)}, S_{i'}^{(t)}) \quad (6)$$

Since $v \in A_i^{(t)}$, and since we have conditioned on the event $\mathcal{F}^{(t)}$ (see Definition 4.8), we get:

$$D_v(Z_i^{(t)}, S_{i'}^{(t)}) > 0 \quad \text{for all } i' \in [i+1, L-1]. \quad (7)$$

Equations (5), (6), (7) imply that the decrease in $\mathcal{B}$ is sufficient to pay for the computation performed.

Case 2: A node $v \in Z_i^{(t)}$ is demoted from level $j > i$ to level i in Steps 09-12 of Algorithm 1.

This can happen only if $v \in B_i^{(t)}$. Let C denote the amount of computation performed during this event. By Claim 4.7, we have

$$C = \sum_{i'=(i+1)}^{(L-1)} O(1 + D_v(Z_i^{(t)}, S_{i'}^{(t)})) \quad (8)$$

Let $\gamma = (1+\epsilon)^4/(1-\epsilon)^2$. Equation (9) holds since $v \in B_i^{(t)}$ and since we conditioned on the event $\mathcal{F}$. Equation (10) follows from equations (8), (9) and the facts that γ, c are constants,

$$D_v(Z_i^{(t)}, S_{i'}^{(t)}) \leq \gamma c \log n \text{ for all } i' \in [i, L-1] \quad (9)$$

$$C = O(L \log n) \quad (10)$$

Let Δ be the net decrease in the overall potential $\mathcal{B}$ due to this event. We make the following observations.

1. By eq. (9), the potential $\Phi(v)$ decreases by at least $(j-i) \cdot (L/\epsilon) \cdot ((1-\epsilon)^2\alpha - \gamma) \cdot (c \log n)$.
2. For $u \in V \setminus \{v\}$ and $i' \in [1, i] \cup [j+1, L-1]$, the potential $\Gamma_{i'}(u)$ remains unchanged. This observation, along with equation (9), implies that the sum $\sum_{u \neq v} \Phi(u)$ increases by at most $(L/\epsilon) \cdot \sum_{i'=(i+1)}^{j} D_v(Z_i^{(t)}, S_{i'}^{(t)}) \leq (j-i) \cdot (L/\epsilon) \cdot (\gamma c \log n)$.
3. For every $i' \in [1, i]$, and $e \in S_{i'}^{(t)}$ the potential $\Psi_{i'}(e)$ remains unchanged. Next, consider any $i' \in [i+1, L-1]$. For each edge $(u, v) \in S_{i'}^{(t)}$ with $u \in Z_i^{(t)}$, the potential $\Psi_{i'}(u, v)$ increases by at most $3(j-i)$. For every other edge $e \in S_{i'}^{(t)}$, the potential $\Psi_{i'}(e)$ remains unchanged. These observations, along with equation (9), imply that the sum $\sum_{i'} \sum_{e \in S_{i'}} \Psi_{i'}(e)$ increases by at most $\sum_{i'=(i+1)}^{(L-1)} 3(j-i) \cdot D_v(Z_i^{(t)}, S_{i'}^{(t)}) \leq (j-i) \cdot (3L) \cdot (\gamma c \log n)$.

Taking into account all these observations, we get:

$$\begin{aligned} \Delta &\geq (j-i)(L/\epsilon)((1-\epsilon)^2\alpha - \gamma)(c \log n) \\ &\quad -(j-i)(L/\epsilon)(\gamma c \log n) - (j-i)(3L)(\gamma c \log n) \\ &= (j-i) \cdot (L/\epsilon) \cdot ((1-\epsilon)^2\alpha - 2\gamma - 3\epsilon\gamma) \cdot (c \log n) \\ &\geq Lc \log n \end{aligned} \quad (11)$$

The last inequality holds since $(j-i) \geq 1$ and $\alpha \geq (\epsilon + (2+3\epsilon)\gamma)/(1-\epsilon)^2 = 2 + \Theta(\epsilon)$, for some sufficiently small constant $\epsilon \in (0, 1)$. From eq. (10) and (11), we conclude that the net decrease in the overall potential $\mathcal{B}$ is sufficient to pay for the cost of the computation performed.

5. OPEN PROBLEMS

An obvious question is whether the $(4+\epsilon)$ approximation ratio provided by our algorithm is tight. In particular, it will be interesting if one can improve the approximation ratio to $(2+\epsilon)$ to match the case where an update time is not a concern. Getting this approximation ratio even with larger space complexity is still interesting. (Epasto et al. [15] almost achieved this except that they have to assume that the deletions happen uniformly at random.) It is equally interesting to show a hardness result. Currently, there is only a hardness result for maintaining the optimal solution [23]. It will be interesting to show a hardness result for approximation algorithms. Another interesting question is whether a similar result to ours can be achieved with polylogarithmic *worst-case* update time. Finally, a more general question is whether one can obtain space- and time-efficient fully-dynamic algorithm like ours for other fundamental graph problems, e.g. maximum matching and single-source shortest paths.

Acknowledgements

The research leading to these results has received funding from the European Unions Seventh Framework Programme (FP7/2007-2013) under grant agreement 317532 and from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Grant Agreement number 340506. Part of this work was done while S. Bhattacharya and D. Nanongkai were in Faculty of Computer Science, University of Vienna, Austria.

6. REFERENCES

[1] K. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 459–467, 2012.

[2] K. Ahn, S. Guha, and A. McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Symposium on Principles of Database Systems (PODS)* , pages 5–14, 2012.

[3] K. Ahn, S. Guha, and A. McGregor. Spectral sparsification in dynamic graph streams. In *APPROX-RANDOM*, pages 1–10, 2013.

[4] A. Angel, N. Koudas, N. Sarkas, and D. Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *PVLDB*, 5(6):574–585, 2012.

[5] B. Bahmani, A. Goel, and K. Munagala. Efficient primal-dual algorithms for MapReduce. In *Algorithms and Models for the Web Graph (WAW)* , 2014.

[6] B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and mapreduce. *PVLDB*, 5(5):454–465, 2012.

[7] S. Bhattacharya, M. Henzinger, D. Nanongkai, C. E. Tsourakakis. Space- and Time-Efficient Algorithms for Maintaining Dense Subgraphs on One-Pass Dynamic Streams. `http://arxiv.org/abs/1504.02268`

[8] G.S. Brodal and R. Fagerberg. Dynamic representation of sparse graphs. In *Workshop on Algorithms And Data Structures, (WADS)*, pages 342–351, 1999.

[9] A. Chakrabarti, S. Khot, and X. Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003)*, pages 107–117, 2003.

[10] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In Klaus Jansen and Samir Khuller, editors, *APPROX*, volume 1913 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2000.

[11] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM J. Comput.*, 34(6):1370–1379, 2005.

[12] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. Optimal sampling from distributed streams. In *Symposium on Principles of Database Systems (PODS)*, pages 77–86, 2010.

[13] A. Czumaj and C. Sohler. Sublinear-time algorithms. In *Property Testing*, pages 41–64, 2010.

[14] Y. Dourisboure, F. Geraci, and M. Pellegrini. Extraction and classification of dense communities in the web. In *Proceedings of the 16th international conference on World Wide Web (WWW)*, pages 461–470, 2007.

[15] A. Epasto, S. Lattanzi, and M. Sozio. Efficient densest subgraph computation in evolving graphs. In *Proceedings of the 24th International World Wide Web Conference (WWW)* (to appear).

[16] P. Erdös, A. Stone, et al. On the structure of linear graphs. *Bull. Amer. Math. Soc*, 52:1087–1091, 1946.

[17] S. Even and Y. Shiloach. An On-Line Edge-Deletion Problem. *Journal of the ACM*, 28(1):1–4, 1981.

[18] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB*, pages 721–732, 2005.

[19] A. Goel, M. Kapralov, and S. Khanna. Perfect matchings in o(nlog n) time in regular bipartite graphs. *SIAM J. Comput.*, 42(3):1392–1404, 2013. Announced at STOC 2010.

[20] A. V. Goldberg. Finding a maximum density subgraph. Technical report, Berkeley, CA, USA, 1984.

[21] O. Goldreich. A brief introduction to property testing. In *Studies in Complexity and Cryptography*, pages 465–469, 2011.

[22] O. Goldreich. Introduction to testing graph properties. In *Studies in Complexity and Cryptography*, pages 470–506, 2011.

[23] M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *47th ACM Symposium on Theory of Computing (STOC)*, 2015.

[24] H. Jowhari, M. Saglam, and G.Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Symposium on Principles of Database Systems (PODS)*, pages 49–58, 2011.

[25] M. Kapralov, Y.T. Lee, C. Musco, C. Musco, and A. Sidford. Single pass spectral sparsification in dynamic streams. In *55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 561–570, 2014.

[26] M. Kapralov and D. Woodruff. Spanners and sparsifiers in dynamic streams. In *ACM Symposium on Principles of Distributed Computing, (PODC)*, pages 272–281, 2014.

[27] B. Kapron, V. King, and B. Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 1131–1142, 2013.

[28] E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Saunders College Publishing, Fort Worth, 1976.

[29] V. Lee, N. Ruan, R. Jin, and C Aggarwal. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*, pages 303–336. 2010.

[30] D. Matula. A min-max theorem for graphs with application to graph coloring. In *SIAM Review*, volume 10, page 481, 1968.

[31] M. Monemizadeh and D. Woodruff. 1-pass relative-error l_p-sampling with applications. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1143–1160, 2010.

[32] I. Munro and M. Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980. Announced at FOCS'78.

[33] K. Onak. Sublinear graph approximation algorithms. In *Property Testing*, pages 158–166, 2010.

[34] A. Pagh and R. Pagh. Uniform hashing in constant time and optimal space. *SIAM J. Comput.*, 38(1):85–96, 2008.

[35] D. Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2009.

[36] R. Rubinfeld and A. Shapira. Sublinear time algorithms. *SIAM J. Discrete Math.*, 25(4):1562–1588, 2011.

[37] A. Erdem Sariyüce, B. Gedik, G. Jacques-Silva, K. Wu, and Ü. Çatalyürek. Streaming algorithms for k-core decomposition. *PVLDB*, 6(6):433–444, 2013.

[38] G. Szekeres and H. S Wilf. An inequality for the chromatic number of a graph. *Journal of Combinatorial Theory*, 4(1):1–3, 1968.

[39] L. Tang and H. Liu. Graph mining applications to social network analysis. In *Managing and Mining Graph Data*, pages 487–513. 2010.

[40] C E. Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the 24th International World Wide Web Conference (WWW)*, 2015 (to appear).

[41] C E. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, M. Tsiarli. Denser Than the Densest Subgraph: Extracting Optimal Quasi-cliques with Quality Guarantees In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 104–112, 2013.